$2.50

68

Popular Computing

The magazine for those interested in the art of computing.

November 1978 Volume 6 Number 11

7	2	9	9	4	3	5	8	2	6	3	0	1	0	8	6	1	3	4	3	3
8	4	6	8	8	7	4	2	8	2	9	7	7	8	8	3	0	2	7	4	3
1	4	9	2	5	4	2	7	2	3	3	7	4	3	4	8	6	9	3	9	6
8	9	3	3	2	0	8	0	5	2	4	6	6	8	2	7	0	9	8	3	1
9	6	1	6	4	4	7	3	7	5	3	6	1	1	0	2	3	0	0	0	6
6	6	2	8	9	9	6	3	3	5	7	5	4	3	7	8	7	3	8	9	2
3	4	4	0	7	1	9	1	1	5	5	4	9	3	5	6	5	2	3	2	7
1	6	4	7	1	0	1	3	8	0	2	8	7	9	8	6	8	6	3	9	1
8	8	1	0	4	0	9	1	7	8	3	4	6	0	5	1	4	3	2	1	8
2	8	7	2	5	3	5	4	8	3	2	2	5	1	8	0	5	5	8	3	3
6	4	1	1	3	8	3	9	5	4	7	3	0	9	0	3	9	6	1	3	7
7	0	5	2	1	2	2	4	4	2	6	8	4	3	4	1	3	6	0	7	8
2	0	7	0	2	8	4	5	1	9	0	1	9	8	4	8	8	6	7	9	4
2	5	2	5	6	2	5	9	8	2	8	0	9	2	7	3	6	2	4	6	9
5	6	4	8	4	2	4	8	6	6	9	2	9	0	4	9	9	8	3	8	2
2	7	8	9	3	6	9	7	5	8	0	6	5	3	6	8	7	3	5	5	5
9	6	2	6	6	7	2	1	3	9	7	8	2	7	3	3	5	0	9	1	0
5	0	1	4	2	9	2	1	0	2	6	4	7	4	4	8	0	6	2	6	9
1	9	8	1	4	1	9	6	4	0	1	1	5	8	8	6	7	3	1	9	3
7	9	7	6	2	9	3	8	6	6	6	5	2	6	6	4	8	4	6	1	5
4	1	3	3	6	0	4	4	5	3	9	8	8	4	0	8	4	4	4	6	3

Minimum Average

Minimum Average

Given the accompanying array of 21 x 21 single-digit numbers. A path is to be traced, starting at the center square (circled) and proceeding to one of the four corners. Movement from square to square is to be diagonally only, never reentering a square and with the path never crossing itself.

The objective is to have the minimum average of the digits; that is, the sum of the digits passed through divided by the number of the squares should be as small as possible. Thus, the path may be any length from 10 squares up to perhaps 400 squares.

The elements in the array are not arbitrary, and may be generated by the following scheme:

$$\text{Set } x_0 = 100$$

$$x_{n+1} = 19x_n \mod 983$$

with the low-order digit at each stage being the digit in our 21 x 21 array. The recursion generates the following sequence:

$x_1 = 917$ giving 7

$x_2 = 712$ 2

$x_3 = 749$ 9

$x_4 = 469$ 9

...and so on.

PROBLEM 249

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

Publisher: **Audrey Gruenberger**
Editor: **Fred Gruenberger**
Associate Editors: **David Babcock**
Irwin Greenwald
Patrick Hall
Contributing Editors: **Richard Andree**
William C. McGee
Thomas R. Parkin
Edward Ryan
Art Director: **John G. Scott**
Business Manager: **Ben Moore**

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each. Copyright 1978 by POPULAR COMPUTING.

ART OF COMPUTING 15

A Sort of Book Review

When man seized the loadstone of
science, the loadstar of superstition
vanished in the clouds.

--W. R. Alger

First there was Strunk's The Elements of Style, published in 1919 and resurrected in 1959 by E. B. White, who wrote a charming New Yorker article about the book and caused Macmillan to republish it. Strunk's little book is worth rereading about once a year, if you care at all about putting English sentences together properly.

Next, the idea of applying Strunk's philosophy to computer programs popped out in The Elements of Fortran Style (Kreitzberg and Schneiderman, 1972).

The peak was reached in 1974 with the publication of The Elements of Programming Style, by Kernighan and Plauger (which we reviewed in our issue No. 13, April, 1974).

All this history is recounted because of two events in publishing: the second edition of the Kernighan and Plauger book, and the appearance of a new applicant to the club:

The Little Book of BASIC Style

...John M. Nevison
Addison-Wesley 1978, 151 pages,
6 x 9 paper cover.

One can almost take care of Kernighan and Plauger's second edition in one word: Hooray. It is pleasant to see that one of the ten best books ever written about computing is doing well. The second edition has added some material, as the authors explain:

> We have revised The Elements of Programming Style extensively... The first edition avoided any direct mention of the term "structured programming," to steer well clear of the religious debates then prevalent. Now that the fervor has subsided, we feel comfortable in discussing structured coding techniques that actually work well in practice.
>
> The second edition devotes a whole new chapter to program structure, showing how top-down design can lead to better organized programs. Design issues are discussed throughout the text. We have made considerable use of pseudo-code as a program development tool.

But otherwise it's much the same excellent book, chock full of distilled wisdom, in a form that is easy to understand and hard to forget, such as:

Make it right before you make it faster.

Keep it right when you make it faster.

Make it clear before you make it faster.

Don't patch bad code--rewrite it.

But now we come to The Little Book of BASIC Style, one of the Addison-Wesley series in the Joy of Computing. There is little joy in watching an amateur feebly trying to imitate the masters.

Let's begin with the quote: "Anyone who has written and run a computer program is literate in computing" (emphasis his). Not in my book. To paraphrase that bit of wisdom: "Any freshman who has written and typed a theme is literate in English."

Nevison's book has a lot of rules, just like the other books, but these are not distilled wisdom and they make little sense by themselves. Consider these:

Nest structures that work together.

Make the line flow left to right.

Match variables to ideas.

Nevison has evidently written many small programs in BASIC; he has discovered that indentation in code makes it more readable. He has embraced several fetishes that are only annoying to others (such as constant allusions to the "REM-5 convention" which is never defined or explained; and insistence that all statement numbers contain the same number of digits) and evidently believes that formal copying of a good book (with added badly-drawn cartoons) makes a new good book. Anyone who writes code with jumps to non-existent line numbers (or, worse, with jumps to REM lines) should not be giving advice about style.

I am being rough with Mr. Nevison, perhaps excessively so. His book will probably not hurt anyone, and might even encourage some students to think about writing better programs. However, any random page of Kernighan and Plauger is worth any chapter of Nevison in terms of usable, practical advice.

In his Preface, Nevison says:

> You may well ask "Why follow any rules?" The answer is quite simple: because they will help you write a better program. A better program is one that is more likely to be correct, one that is easier to read, and one that is simple to use. For a number of years flow charts have been advocated as a way to help explain programs, but now there is evidence that they do not help (Schneiderman, 1977).

The reference is to the article "Experimental Investigations of the Utility of Detailed Flowcharts in Programming" by Ben Schneiderman (co-author of that 1972 style book), Richard Mayer, Don McKay, and Peter Heller, in the June 1977 Communications of the ACM.

Schneiderman and his group at Indiana designed some experiments to investigate the usefulness of flowcharting as a tool of program design. Their description of one experiment follows:

> Experiment I (Composition)
>
> This first experiment was designed to study how the creation of a detailed flowchart assisted the subjects in composing a program. Much of the literature on flowcharting claims that it is most helpful as a program design aid which clarifies the problem to the programmer.

Subjects. The subjects were students in an introductory computer programming course using Fortran. The textbook used flowcharts to illustrate program development and flowcharts were used by the instructor. The experiment was conducted by including the materials as part of the second of three in-class examinations which constituted the major part of the course grade.

Procedure and materials. Thirty-four subjects (flowchart group), received test instructions which indicated that they were to write a flowchart and then a program for a given problem. The flowchart counted for 15 points, the program 25 points. Twenty-eight subjects (nonflowchart group) were instructed to merely write the program, which counted for the full 40 points. The subjects were given as much time as they wanted to complete the test. The grading was done by a graduate student with much experience in grading programs and in consulting with students about programs. The results were returned to the students at the next meeting of the class.

Results

The scores on the program composition task were normalized to 100 percent. The flowchart group mean score was 94 while the nonflowchart group mean was 95. A t-test showed no significant difference between the two groups. The flowchart group had a mean score of 13.1 on the flowchart or 87.3 out of 100.

Discussion

The requirement to produce a flowchart seemed to have no benefit or harm on the subjects' ability to prepare a program described by that flowchart. This was in spite of the fact that the problem had been chosen to favor the flowchart group by having a relatively complicated branching pattern. The relatively good scores indicate that all the subjects found the task to be straightforward.

They had heard of controlled experiments, and they wanted the mantle of science wrapped around their project. Unfortunately, assuming one or two of the attributes of science does not make anyone a scientist. In this case, the whole affair is not only non-science, but sheer nonsense. You don't have a controlled experiment by setting up two groups, giving one special treatment, and then giving a test. In a *controlled* experiment, the effects of *all* extraneous variables are eliminated or cancelled out.

Just what was the flowchart group taught? And by whom? Using what text? What was the problem that was "chosen to favor the flowchart group"? If the problem had any kind of complicated logic, how did the nonflowchart group go about it? Did anyone ask? Did the instructor for the flowchart group know anything about flowcharting from personal use, or was his knowledge gained from textbooks? If anyone involved in the experiment had acquired his knowledge from the existing texts on flowcharting, they would know virtually nothing about the subject. They would be adept at the use of some 40 standard symbols (and even, in one case, have been told the proper angle to hold their pencil in using a plastic template), but they would not have been told what to put in the boxes, or why. They would have been exposed to many diverse sloppy habits, and to almost no real problem solutions in which flowcharts might help in getting one's thinking organized.

I hold no special brief for flowcharts as the way to organize a problem solution, or as the tool for communicating such a solution to others and to yourself (perhaps a year later), or as the way of documenting a solution, or as the device for helping to maintain a program. There are many ways to go at it, and some people can program perfectly without any of them. I do submit that flowcharts are a way, and one that helps muddled minds like mine to get their thinking squared away. In none of their work did the Schneiderman group test for program maintenance, or for having the original programmer recall what he did a year back. They seem to feel that ease of coding is the paramount, if not the only, virtue. The whole thing is ridiculous. "Research" work like this should be laughed away, and would be if people could only think a little. With this kind of "science" one can prove that pi is a rational number, that ESP exists, and the efficacy of the tooth fairy--as many people have done. Our journals are cluttered with such "studies." It beats working for a living, and it helps guarantee promotion. W. R. Alger was dead wrong.

In 1972, a symposium was held at Anaheim (reported in our issue No. 9) in which one of the subjects discussed was the use of flowcharts as a tool not only of program design but one of communication and documentation. To provide a takeoff point for the discussion at that meeting each attendee was given, a month in advance, a problem situation to be flowcharted. Each of the dozen attendees dutifully produced a flowchart. The original idea was to circulate copies of these flowcharts, with an eye toward improving the technique.

That minor experiment was a total disaster. Of the twelve flowcharts:

(a) At least half of them patently did not solve the given problem.

(b) No two of them used even remotely similar notation.

(c) None of them used ANSI standard flowchart notation.

(d) For at least half of the flowcharts, a group of outside experts could not deduce what action was intended over most of the flowchart.

(e) The overall physical size of the flowcharts varied by a factor of five, but with a variation in the size of the flowchart symbols by a factor of only two.

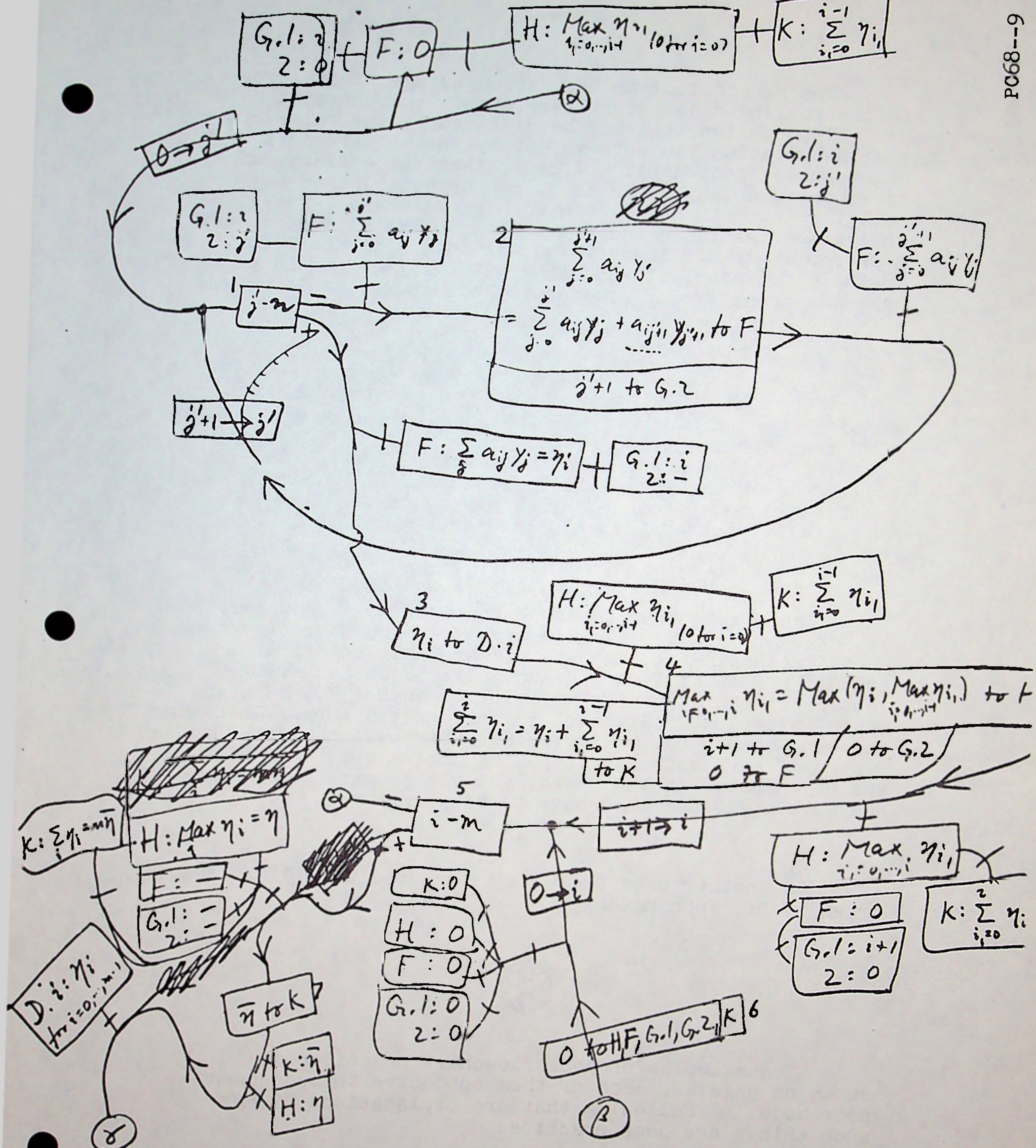
(f) Exactly one of the flowcharts was clearly logical and correct; the author had coded the problem solution from his flowchart and had tested the code adequately.

These people were experts; in fact, senior citizens of our field. To be sure, some of them disliked the use of flowcharts and had participated in the experiment only out of courtesy.

One reason that flowcharts are in disrepute is simply that they have never been used properly, and certainly their use has not been taught by anyone who knows how to use them. If you let students (and instructors) go wild and draw anything that they think is a flowchart, then you will get chaos, which is precisely what we now have. One has only to examine any introductory text to see the awful state of affairs in this area. (Of course, most texts pay effusive tribute to the need, utility, and desirability of flowcharting, and then ignore it for the rest of the book.)

Do you need more convincing? The addendum to this article shows ten examples, taken from textbooks and student papers, each of them illustrating some way NOT to flowchart. Try your hand at them.

What are the logical alternatives to flowcharts? HOw do ordinary programmers organize their thinking? I can understand how masters at programming (by which I mean the top 20 active programmers in the country, for example) are able to by-pass the flowcharting stage, and thus treat it with contempt. How are the rest of us to design complex programs, maintain them, document them, and communicate our reasoning to others?



G.1: i 2: 0
F: 0
H: Max η_{i_1} $i_1=0,\dots,i-1$ (0 for i=0)
K: $\sum_{i_1=0}^{i-1} \eta_{i_1}$
α
0 → j'
G.1: i 2: j'
F: $\sum_{j=0}^{j'} a_{ij} y_j$
2
$\sum_{j=0}^{j'+1} a_{ij} y_j = \sum_{j=0}^{j'} a_{ij} y_j + a_{ij'+1} y_{j'+1}$ to F
j'+1 to G.2
G.1: i 2: j'
F: $\sum_{j=0}^{j'+1} a_{ij} y_j$
1
j' - n
j'+1 → j'
F: $\sum_j a_{ij} y_j = \eta_i$
G.1: i 2: —
3
η_i to D.i
H: Max η_{i_1} $i_1=0,\dots,i-1$ (0 for i=0)
K: $\sum_{i_1=0}^{i-1} \eta_{i_1}$
4
Max$_{i_1=0,\dots,i}$ η_{i_1} = Max(η_i, Max$_{i_1=0,\dots,i-1}$ η_{i_1}) to H
$\sum_{i_1=0}^{i} \eta_{i_1} = \eta_i + \sum_{i_1=0}^{i-1} \eta_{i_1}$ to K
i+1 to G.1 / 0 to G.2
0 to F
5
i - m
i+1 → i
0 → i
K: $\sum_i \eta_i = m\bar{\eta}$
H: Max $\eta_i = \eta$
F: —
G.1: — 2: —
D.i: η_i for i=0,…,m-1
$\bar{\eta}$ to K
K: $\bar{\eta}$
H: η
K: 0
H: 0
F: 0
G.1: 0 2: 0
0 to H, F, G.1, G.2, K
6
β
γ
H: Max η_{i_1} $i_1=0,\dots,i$
F: 0
G.1: i+1 2: 0
K: $\sum_{i_1=0}^{i} \eta_i$

John von Neumann's November, 1946 flowchart.

The proper tool to use, at any stage of the process of controlling a computer, is probably the one that helps you (although the tool may be dictated by the rules of the installation you work in, or by the requirements of a government contract). I doubt that the efficacy of flowcharts, or any other tool, can be established by childish "experiments," even when they invoke t-tests and analysis of variance and try to sound like science at work. A proper study of flowcharting vs. some other tool (or, as in the cited case, flowcharting vs. nothing) would entail hundreds of different students, thousands of dollars, and a year's hard work.

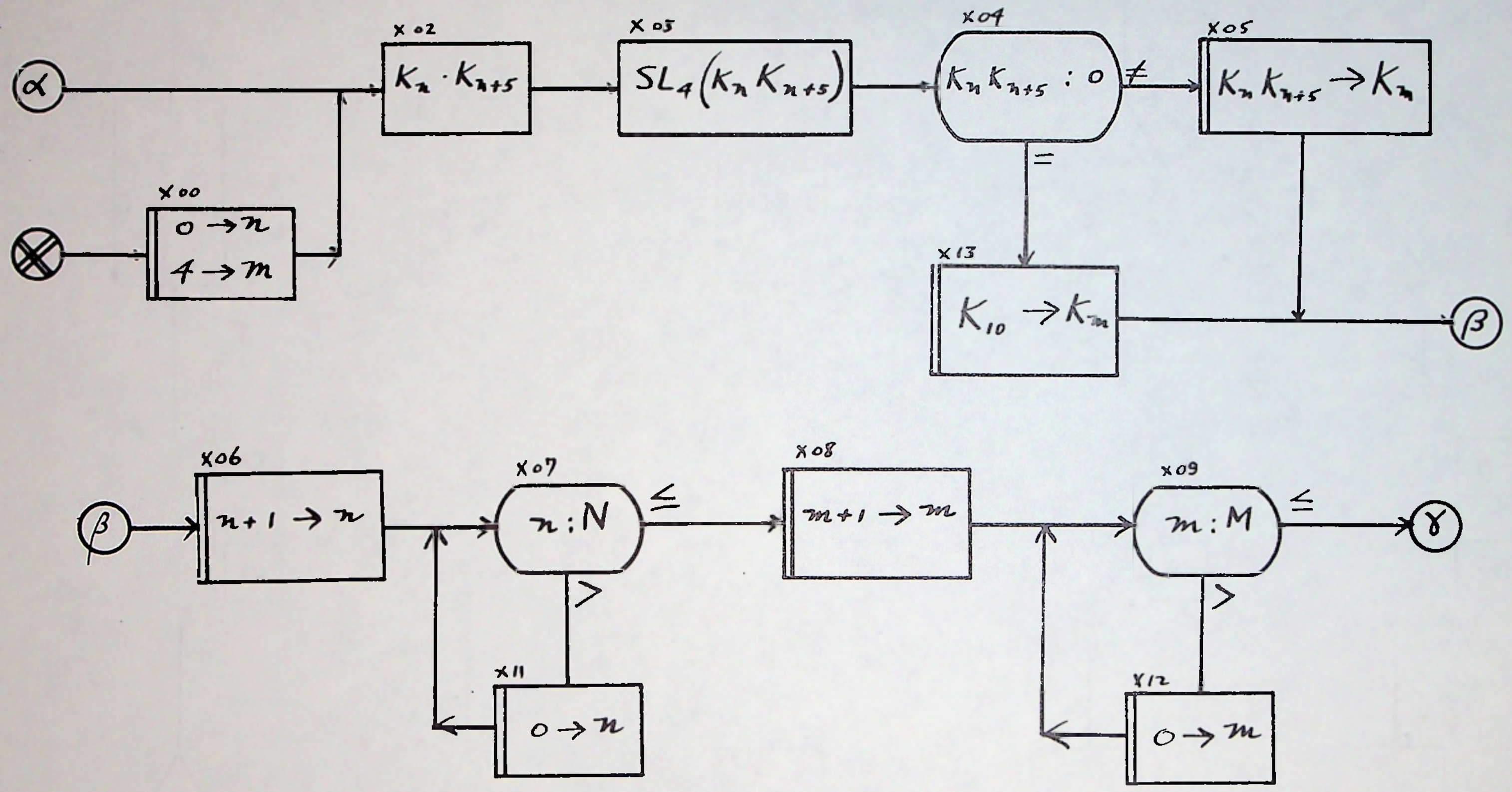
Some kind of tool is needed for the various tasks associated with programming; that tool might be the pseudo-code that Kernighan and Plauger refer to. It is quite likely that flowcharts cannot function for all tasks for all people--probably no single tool can. Whatever tool is tried as a replacement will have to be designed and taught a lot better than flowcharts ever were.

There are other tools, to be sure, but flowcharting has a long and honorable history. There is reproduced on the accompanying page what is probably the first computing flowchart ever drawn. It is copied from a letter written by von Neumann in November 1946, some five months after publication of the celebrated Burks, Goldstine, and von Neumann paper "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument." There was no computer in the world, but von Neumann was describing how such a device would have to be programmed.

On another page there is reproduced a 1949 flowchart drawn by Dr. John Mauchly.

Ten examples of poor flowchart practices are shown on page 12, each of them conducive to subsequent poor code. Following that are explanations of why such things are poor practice.

In nearly every other discipline, we seem to insist on correct, standard practices from beginners, as a necessary prelude to competent work. Why is it that we permit sloppy workmanship and permissiveness in our field?

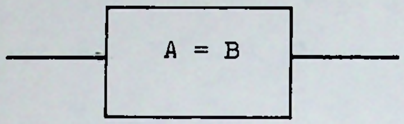


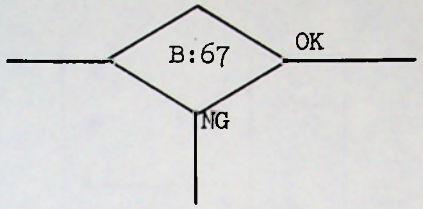
$K_m = K_n \cdot K_{n+5}$ or K_{10}

n = running index of factors; N=4
m = running index of product; M=9
SL_4 = shift left 4 positions
α = point of entry to routine
γ = point of exit from routine

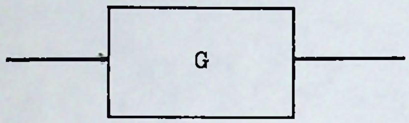
from the paper
"Pseudo-Random Numbers"
Dr. John W. Mauchly
December 29, 1949

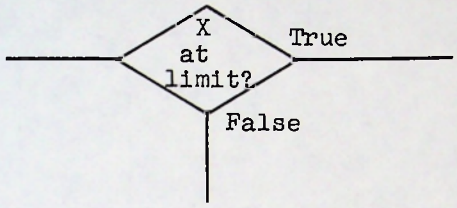
1 6



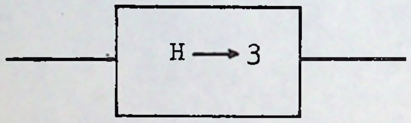


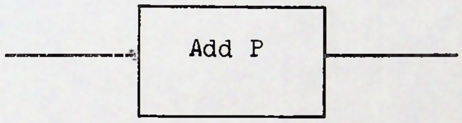
2 7



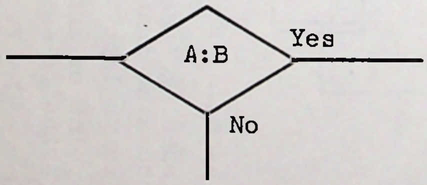


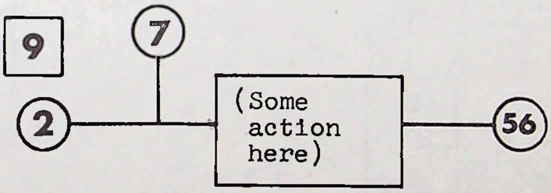
3 8



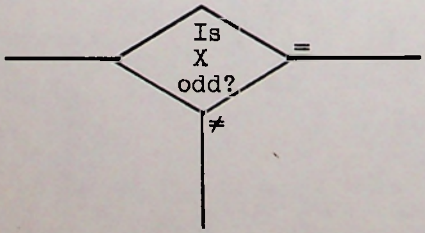


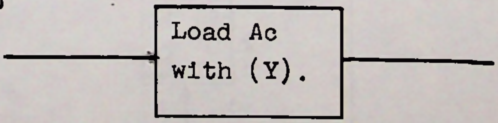
4 9





5 10





$\boxed{1}$ This is ambiguous. Is A to be made equal to B, or is B to be made equal to A, or are they simply equal to each other now? If B is to be given the value of A, then the notation:

$$A \longrightarrow B$$

("A replaces B") says what is to be done, and there can be no confusion. The sloppy notation shown in the first example comes about by permitting things like:

$$X = X + 1$$

(a Fortran-type statement), which is <u>coding</u> and has no place on flowcharts.

$\boxed{2}$ This is one of those things that were all clear to the person who wrote it at the time he wrote it. Now it makes no sense; no action is called for.

$\boxed{3}$ The right pointing arrow notation stands for "replaces," as we have noted. So what could this possibly mean? The value of variable H replaces word 00003 of storage? If variable H is to be given the value 3 (which is what is usually meant) then the arrow points to the H:

$$3 \longrightarrow H$$

or

$$H \longleftarrow 3$$

(the latter being rather awkward English, "H is replaced by 3," but this is preferred by many, particularly as it translates directly into Fortran code.)

$\boxed{4}$ Inside the box is the command "Compare A to B." Possible outcomes are: equal and not equal; greater than, equal to, and less than. "Yes" and "No" are not English responses and make little sense, no matter how clear the intention is to the writer.

$\boxed{5}$ Again, we have an English question, and the only possible outcomes are "Yes" and "No." Nitpicking? Perhaps--but why not be neat and correct?

$\boxed{6}$ More bad English.

$\boxed{7}$ Proper responses here would again be "Yes" and "No," but "True" and "False" are not far from correct. The trouble is, as soon as you permit this, things can get out of hand, and the whole world becomes "True" and""False." (There are computing instructors who insist that <u>all</u> decisions be reduced to True and False, thus adding to computing illiteracy.)

$\boxed{8}$ Good grief--add P to what?

$\boxed{9}$ It's either Reference 2 <u>or</u> it's Reference 7, but why would it ever be both? This indicates muddled thinking, which is something that good flowcharts are supposed to clear up. At the other end, there is nothing logically wrong with Reference 56, but it's probably just cute. When there are more than ten Reference points on one flowchart, something is very likely wrong. And if there are ten or less Reference points, then the use of "56" is simply coy.

$\boxed{10}$ This is pure coding--it can have little or nothing to do with the <u>solution</u> of a problem.

Fred Gruenberger

A Fortran Coloring Book

by Dr. Roger Kaufman
M.I.T. Press 1978, 285 pages 6 x 9, $6.95

Donald Alcock started something, with his book Illustrating BASIC in 1977. He produced the entire contents of the book himself, hand lettered, including the title page and the copyright notice. It was done with charm, flair, and good taste.

If a little is good, a lot might be better.

Perhaps. Dr. Kaufman's Fortran text uses the same general idea (hand lettered charts), but laced with slightly corny humor, as for example (page 237):

Here's a Problem that Gauss Against the Grain; Seidel up to it slowly!

This may or may not make Fortran palatable to young people (assuming, of course, that the book is aimed at young people); it might depend on how much you can appreciate puns on the Gauss-Seidel method of solving simultaneous linear equations. One suspects that in Dr. Kaufman's own hands at George Washington University, the light treatment makes his Fortran course the delight of the freshman class. It could also be a delight for someone who has access to a Fortran compiler and wishes to learn on his own. At the least, this must be the least ponderous Fortran text book ever written.

Will the text work, then, for a class taught by someone else? That remains to be seen. The Fortran material appears to be impeccable, and Dr. Kaufman (who is well grounded in both mathematics and computing) believes in teaching much about computing along with the rules of Fortran. He is adept at making complex ideas clear, and he is generally literate and witty. The question remaining is, how much wit like the following do you want in a textbook?:

Now we all know that FORTRAN is irrational, and Bolzano's bambino would have wet his diaper if he knew that computerniks would someday use the word "real" to mean "numbers with decimal points," i.e., floating point numbers. Thus, the word "real" is a real bummer because it has so many meanings it can make you reel. Usually what is meant is ~~to~~- perfectly clear from the context, however.

homework 6

INTERSECTION...A Flowchart Exercise

We have two arrays of numbers in storage:

Array A, for which the number of elements in the array is given by the contents of word AL; and

Array B, for which the number of elements in the array is given by the contents of word BL.

The numbers in array A are all unique; that is, there are no duplicates within the array. Similarly, there are no duplicates within array B. That is all we know about the numbers in the two arrays.

We want to produce a printed list of all the numbers that are common to both arrays. Each number in one array will have to be compared to every number in the other array; this will require a loop within a loop.

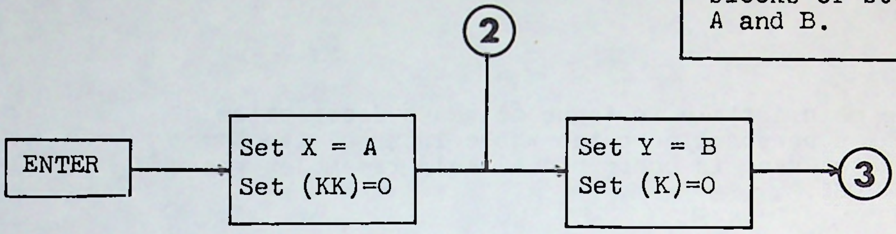
Draw a flowchart for this logic.

PROBLEM **250**

Note to instructors: A possible solution is given on the next page...

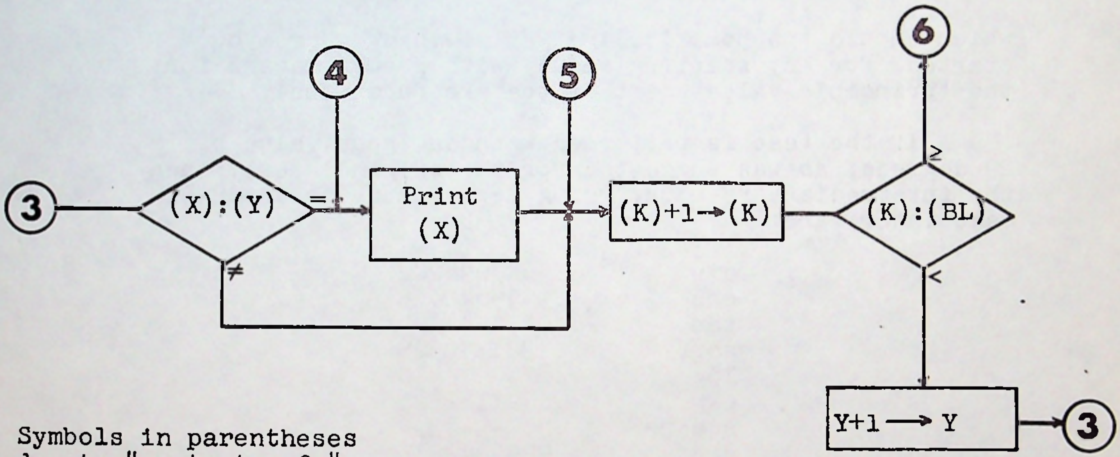
Finding and printing the numbers common to two blocks of storage, A and B.

PC68-17

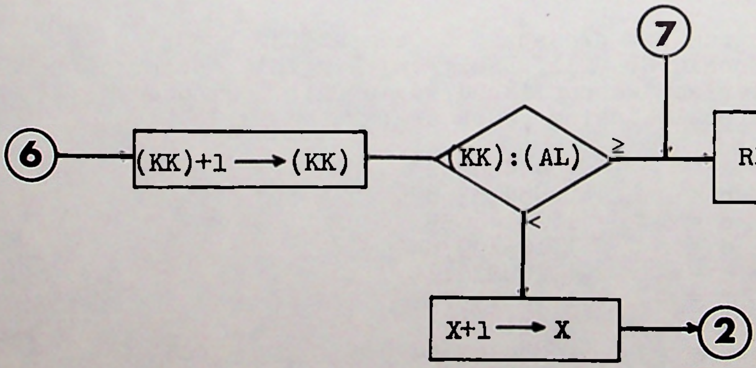


Counter K is for the inner loop, to count through array B.

Counter KK is for the outer loop, to count through array A.



Symbols in parentheses denote "contents of." All others are addresses.



feedback

The main article in issue 65 was a description of PeCos One, a personal computer whose internal language as far as the user is concerned closely resembles the language JOSS.

To demonstrate the quality of the arithmetic and function subroutines that were written for PeCos, we repeated a test (from issue 46) that was devised for desk calculators; namely, to carry out this sequence of calculations:

sin cos tan sqrt log exp square arctan arccos arcsin

which should (mathematically) get you back where you started, for any starting value (with proper regard for the "principle value" part of the arc functions).

If the test is performed with an input value of 29 degrees, as was suggested for the original test, then the intermediate results, using degree mode on an SR-52 calculator, are:

sin	.4848096202
cos	.9999642015
tan	.0174544399
sqrt	.1321152525
log	-.8790470410
exp	.1321152525
square	.0174544399
arctan	.9999642015
arccos	.4848098549
arcsin	29.00001537

A short program was presented to repeat this test on PeCos One. Donald C. Hill, Dahlgren, Virginia, pointed out that the comparison we published was unfair. Indeed, the intermediate results using PeCos are:

```
a =     .48480962
b =     .884763689
c =    1.22146655
d =    1.10519978
e =     .100026115
f =    1.10519975
g =    1.22146649
h =     .884763664
j =     .484809677
k =     .506145549
```

(for an input of .506145483, which is the radian equivalent of 29 degrees.)

Mr. Hill points out:

> The test is an unfair comparison between PeCos and calculators because PeCos performed the test in radians while the calculators performed the test in degrees. The second step of the PeCos version is to take the cosine of .4848... radians, while the second step of the calculator version is to take the cosine of .4848... degrees.
>
> Thus, they are not performing the same test. Try performing the test on a calculator in radians. My Sharp ELSI MATE EL-5805 gets exactly the correct answer. My HP's do better than PeCos.
>
> The major difference between degrees and radians is at the arccos step where the PeCos version is finding the arccos of .8847..., and the calculators are finding the arccos of .999964... At that point, every unit that the calculator input is off results in about 6800 units of error in the output. Hence the calculators are bound to lose about 4 digits of accuracy.
>
> Nobody expects 8-bit trig functions to do as well as calculators and there is no reason that they should.

Fair enough. Mr. Hill suggested running the test on a calculator in radian mode. Again using .5061454831 as the starting value, the sequence of results comes out:

sin	.4848096203
cos	.8847636897
tan	1.221466558
sqrt	1.105199782
log	.1000261167
exp	1.105199782
square	1.221466558
arctan	.8847636897
arccos	.4848096203
arcsin	.5061454831

and the last value, translated back to degrees, comes to 29.00000000.

So your program works... but

working is not sufficient!

Is it easy to read and understand?

Is it easy to describe or explain?

Is it easy to improve or optimize?

Is it easy to code (implement) and document?

Is it easy to test or debug?

Is it easy to extend or modify?

Is it easy to analyse and evaluate?

Is it easy to prove?

Structured programs created top-down are easier!

John Motil